

Efficient Content-Based Detection of Zero-Day Worms

P. Akritidis*, K. Anagnostakis[†], E. P. Markatos*

*Institute of Computer Science
Foundation for Research & Technology Hellas
P.O. Box 1385 Heraklio, GR-711-10 GREECE
Email: {akritid,markatos}@ics.forth.gr

[†]Distributed Systems Laboratory
CIS Department, Univ. of Pennsylvania
200 S. 33rd Street, Phila, PA 19104, USA
Email: anagnost@dsl.cis.upenn.edu

Abstract—Recent cybersecurity incidents suggest that Internet worms can spread so fast that in-time human-mediated reaction is not possible, and therefore initial response to cyberattacks has to be automated. The first step towards combating new unknown worms is to be able to detect and identify them at the first stages of their spread. In this paper, we present a novel method for detecting new worms based on identifying similar packet contents directed to multiple destination hosts. We evaluate our method using real traffic traces that contain real worms. Our results suggest that our approach is able to identify novel worms while at the same time the generated false alarms reach as low as zero percent.

Index Terms—Security, network-level intrusion detection, Internet worm detection.

I. INTRODUCTION

Recent cyberattack outbreaks have shown that Internet worms are able to infect tens of thousands of Internet computers in less than one hour. For example, the Witty Worm was able to infect more than 20,000 victim computers in less than 60 minutes [1]. Similarly, the Sapphire/Slammer worm was able to infect more than 70,000 victim computers in less than 15 minutes [2]. To make matters worse, theoretic results suggest that well-prepared worms can spread even faster than the above examples, infecting the majority of their victim population in less than 10 minutes [3], [4].

Fortunately, the current generation of Internet worms can be effectively blocked at the network level using filtering mechanisms, such as those employed by firewalls and Intrusion Prevention Systems. These mechanisms are usually based on rules or signatures that describe the worms. However, the generation of those signatures currently requires considerable human effort and consequently takes a lot of time. The current practice suggests that signatures for firewalls and Intrusion Prevention Systems are usually available several hours after the initial outbreak of the worm, which implies that at the current speeds of worm spread, signatures are available *after* the worm has been spread to the majority of its victims. In order to reduce the cost of each worm outbreak, we need to develop methods that are able to detect the worm and generate its signature *before* the worm manages to infect a large percentage of the vulnerable population, i.e. within the first few minutes of its spread. Therefore, it is evident that automatic worm detection and mitigation mechanisms need to be studied.

In this paper we describe a method to detect Internet worms that relies on popular packet payloads and investigate its parameter space with the intention of eliminating false

positives. We show that three parameters, other than mere popularity, play an important role in eliminating false positives: the multiplicity of targeted destinations, the length of the considered content substrings, and their position within their flows.

The rest of the paper is organized as follows: Section II presents the description of our worm detection algorithm, while section III presents its evaluation using realistic network traffic. Section IV places our work in context by contrasting it to related work, and finally section V summarizes and concludes the paper.

II. WORM DETECTION ALGORITHM

In this section we present a worm detection method based on four observations which are commonly found in known worms:

- **Diversity of Destinations:** The network packets that belong to the same worm tend to have a very large number of destinations. Actually, this seems to be an inherent property of all the worms: worms tend to spread to as many victims as possible, and therefore, their network packets seem to have a large number of destinations.
- **Spread by Clients:** Most worms are usually spread by clients, i.e. by computers that initiate a (usually TCP) connection. This property, as well, seems to be an inherent property of the aggressive worms. Indeed, in order for a worm to spread fast, it needs to initiate connections to its potential victims, rather than to wait for the potential victims to connect to it.
- **Payload Repetition:** Several of the network packets that belong to the same worm, tend to contain similar (if not identical) payloads.¹
- **Small Size:** Worms tend to be small in size, in order to spread as fast as possible. A large worm size would prolong infection time and consume bandwidth that could be used for infecting other targets.

Well-known worms such as CODE-RED, Blaster and Welchia, Sapphire, and the Witty worm, depicted all the above four properties.

Thus, to identify new worms, our detection scheme identifies common substrings that appear in the payloads of several (client) packets, which are heading for lots of different destinations. We capitalize on the small worm size observation by

¹It has been proposed that future worms will be polymorphic and will be able to change the payload of the network packets that carry the worm. The detection of such worms is outside the scope of this paper.

```

for each reassembled packet in trace
  for each fingerprint in packet
    if fingerprint in cache
      if packet destination not recorded
        mark as new
        record destination
        if destinations > threshold
          report
    else
      create new entry for fingerprint
      record destination

```

Fig. 1. Pseudocode for worm detection algorithm without the fingerprint selection optimization.

ignoring substrings whose connections have generated large traffic.

Our starting point is the technique presented by Spring and Wetherall [5] for identifying repetitive information transfers using Rabin fingerprints [6], which we appropriately extend for worm detection. We use a cache that holds substrings encountered within a fixed period of time, each cache entry contains an encountered string² together with a list of distinct destinations it was sent to, and the entries are indexed by string.

Our algorithm operates as follows:

- Only traffic from the *initial* part of *client-to-server* flows is processed.
- For each encountered string, if a corresponding cache entry exists and its distinct destination list does not include the current destination, the entry is marked as new and the current destination is recorded in the entry’s distinct destination list. The number of distinct destinations is then compared against a threshold, and an alert is issued if the threshold has been reached.
- Otherwise, if the string is not already in the cache, a new entry is created.
- Finally, old entries are evicted from the cache.

The pseudocode can be found in Figure 1.

A. Repetitive Traffic From Server Replies

Given that rapidly-spreading worms spread mostly through clients, and not through servers, in our implementation we discard server replies and process only client requests, thus preventing content from popular servers from triggering false positives. Our implementation relies on Snort’s [7] session tracking to decide the direction of a packet.

B. Repetitive Packets vs. Repetitive Strings

Many worms are spread using identical packet payloads, and therefore could be easily detected by identifying repetitive packets seen in the network. However, sometimes entire packets may be too coarse-grained for worm detection. For example, the Witty worm [1], has actually implemented random padding of packets. Therefore, in our approach, to identify

²In order to save space the entry contains a 32-bit fingerprint of the string.

payload repetition, instead of entire packets, we consider packet substrings of a fixed length.

C. Stream Reassembly

Clever attackers may easily hide their attack into several different fragmented packets that may be sent out-of-order. To solve this problem, we have used the packet reassembly mechanisms provided by the Snort NIDS [7]. We integrated our filters with Snort in the form of a Snort preprocessor plugin. This way we can take advantage of the existing stream4 preprocessor that comes with Snort and reassembles raw packets into larger ones. The reassembled packets are then fed to the worm detection algorithm.

D. Flow Size Penalty

In this article we have used the crude criterion of ignoring substrings appearing deeper than a fixed offset in their flows. An elaborate worm could exploit this to evade detection by transferring a sufficient amount of junk data before carrying out its attack. However, a refinement of this mechanism that weights each occurrence according to the traffic that has been carried over its connection would transform this abrupt limit into a smooth tradeoff between increased detection delay and decreased worm size.

The main benefit of penalizing strings that appear in large flows is to weaken the impact of peer-to-peer protocol messages sent to multiple destinations over long-lived client connections, preventing them from causing false positives.

E. Performance

For each encountered substring, the system records all the destinations to which it has been sent. Fortunately, most substrings will only be sent to a single destination before they are evicted from the cache and therefore the space required for recording more than one destination does not have to be allocated for the majority of the encountered substrings.

Furthermore, to efficiently compute the hash values of consecutive overlapping packet substrings we employ Rabin fingerprints [6]. The Rabin fingerprint f_α of an n -gram a is computed according to the formula $f_\alpha(a_0, a_1, \dots, a_{n-1}) = \sum_{i=0}^{n-1} a_i \alpha^{n-i-1}$. Rabin fingerprints can be used to incrementally update the hash value of a sliding window over the packet payload, by considering the contribution to the hash of the next byte and removing the contribution to the hash of the last byte of the previous window according to formula $f_\alpha(a_1, a_2, \dots, a_n, a_{n+1}) = \alpha(f_\alpha(a_0, a_1, \dots, a_n) - \alpha^n a_0) + a_{n+1}$. We perform the arithmetic modulo 2^{32} and use a large prime for α .

However, even Rabin fingerprints employ a constant number of operations for each and every byte of network traffic. Given that modern networks deliver up to 10 Gigabytes of traffic per second, even Rabin fingerprints impose a substantial computational overhead to the system. Furthermore, having to account for each and every overlapping substring is impractical. We use two mechanisms to reduce these overheads in our detection scheme:

- Discard server replies
- Fingerprint selection

TABLE I

TOTAL WORM LENGTH, LENGTH OF THE ATTACK PORTION SENT FROM CLIENT TO SERVER, PROTOCOLS, AND TARGETED PORTS FOR VARIOUS WORMS.

Worm	Total Length	Attack Length	Protocol	Dst Port
Witty	600 B + padding	600 Bytes	UDP	random
Sapphire	376 Bytes	376 Bytes	UDP	1434
CodeRedII	3,8KBytes	3,8KBytes	TCP	80
Welchia	10KBytes	1,7KBytes	TCP	135

1) *Discard Server Replies*: We have already mentioned in section II-A that we ignore server replies and focus only on client requests in order to reduce false positives. The same mechanism improves performance as well. Indeed, server replies are typically large contributors to Internet traffic. By focusing our detection algorithm on client requests, instead of server replies, we reduce the load of our detection mechanism by having to compute fewer Rabin fingerprints without reducing its accuracy.

2) *Fingerprint Selection*: It is impractical to process each substring that starts at each and every byte of the network traffic. For example, assume a network packet that is $packetsize$ characters long. Assume also that our algorithm searches for substrings which are n -bytes long. Then, for this specific example, we will end up processing $packetsize - n + 1$ substrings. Given that these substrings are highly *overlapping*, it is possible to reduce the overhead of our approach without reducing its accuracy by employing fingerprint selection, a technique developed by Manber [8] for files and used by Spring and Wetherall [5] for network traffic. Instead of considering all Rabin fingerprints, we sample them based on their value. The amount of samples can be determined by the number of bits set in a sampling mask. A fingerprint is further processed only if the result of applying the mask to it is not zero. A string that matches the sampling criteria is always sampled, so its frequency is not reduced.

Note, that it is theoretically possible for an attacker to exploit this sampling mechanism. Indeed, if the attacker knows the parameters of Rabin fingerprints and the exact value of the mask, (s)he will create worms whose content will never match the mask and therefore will never be sampled. On the other hand, these parameters could vary or the mask bits could be shuffled at regular intervals, so that it would be practically impossible for an attacker to avoid detection by carefully crafting the body of the worm to avoid sampling.

III. EVALUATION

In this section we evaluate the effectiveness and efficiency of our approach and investigate its parameter space using real network traffic traces that contain a real worm.

A. Network Traffic Traces

Two sets of real network traffic traces, gathered from FORTH's Local Area Network in early 2004, have been used for our evaluation. The monitored networks contain about 150 hosts. Each trace is about 5 Gbytes large and contains between 11.7 and 14.4 million network packets that represent traffic

TABLE II

CHARACTERISTICS OF THE TRACES USED IN THE EXPERIMENTS.

Trace	TCP Packets	TCP (Payload) Bytes	Duration	Attacks
TRACE-I	11,746,790	5,108,857,877	2h	102
TRACE-II	14,429,618	5,333,977,518	2h30m	57

from web clients, peer-to-peer programs, SMB shares, IMAP, printers, etc. The traffic characteristics of the traces are shown in table II.

B. Detection

In this section we explore the parameter space of the worm detector that we have described. The parameters of the experiments are:

- substring length,
- distinct destinations threshold,
- substring cache size,
- flow offset limit, and
- sampling mask.

We measure the following quantities:

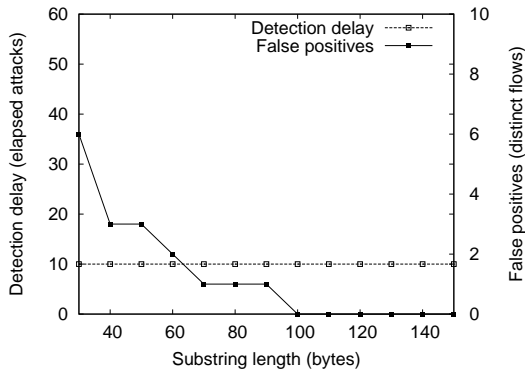
- **false positives**, which are defined to be the number of distinct flows that triggered an alert but did not correspond to any real worm,
- **detection delay**, which is defined as the elapsed worm attacks up to the detection of the worm.

Figure 2 shows the detection delay and the number of false positives incurred by our approach as a function of the substring length for TRACE-I and TRACE-II. We immediately notice that as the substring length increases, the number of false positives decreases. This is as expected. Indeed, it is quite possible for unrelated network packets to contain identical small substrings. Therefore, these identical small substrings that can be found in unrelated (non-worm) network packets, will generate a large number of false alarms. However, as the substring length is getting larger, it is rather unlikely for unrelated network packets to contain large identical substrings. The remaining false positives persist up to a substring length of 150 bytes and are caused by common strings such as protocol headers.

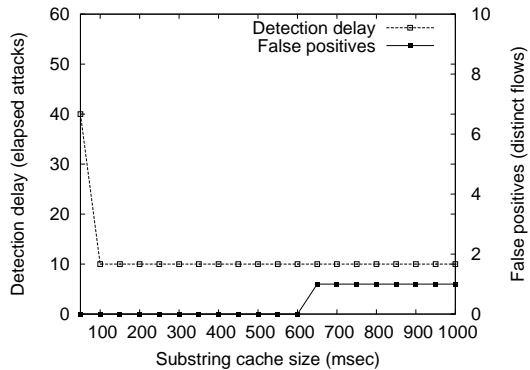
It is very encouraging to see in Figure 2 that as the string length increases beyond 150, the number of false positives reaches zero, which implies that no false alarms for worm outbreaks are generated by our approach. Given that most worms are longer than 150 bytes (as seen in Table I), operating our approach with substring length longer than 150 bytes, will probably be able to identify these known worms without generating any false positives.

We also see that the detection delay is independent of the substring length, and therefore, is plotted as a line parallel to the x -axis. This is because all the true positive alerts were generated by strings belonging to the Welchia worm, which has size larger than 150 bytes.

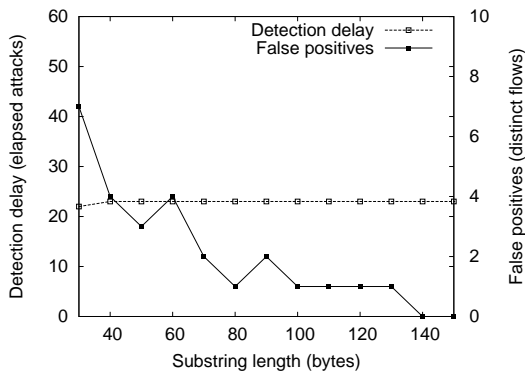
However, the worm contains a few 150-byte strings that can also be found in normal RPC traffic. Filtering these strings would result in inadvertent denial-of-service attacks. This problem is solved by using a string length of 250 bytes or above.



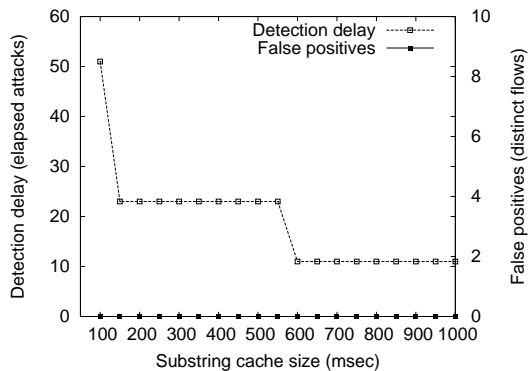
(a) TRACE-I



(a) TRACE-I



(b) TRACE-II



(b) TRACE-II

Fig. 2. Detection delay and false positives as a function of substring length for a fixed cache size of 500 msec, distinct destination threshold of 10, sampling mask value of 0xf, and an offset limit of 100K. We observe that we have zero false positives for substring lengths above 150 bytes, while at the same time we have detection after about 10–20 attacks.

Figure 3 shows the detection delay and the number of false positives as a function of the substring cache size measured in milliseconds. By definition our approach is incapable to detect worms that are encountered less often than the period that substring fingerprints are cached. Indeed, we observe that the worm contained in the traces may evade detection for cache sizes less than 100 msec (Figure 3(b)). On the other hand, we observe that larger cache sizes result in more false positives, as expected, since strings with a lower rate of new destinations are retained in the cache, and can trigger false detection.

Figure 4 shows the detection delay and the number of false positives as a function of the distinct destination threshold. We observe that decreasing the threshold decreases detection delay. This is expected, since less worm attacks are required to trigger detection. However, decreasing the threshold may also result in false positives. This is expected too, since there exist legitimate strings that are sent to more than one destinations.

Finally, in Figure 5 we investigate the effects of various offset limit values. We observe that penalizing strings that appear deep in their flows significantly reduces the encountered false positives. Inspection of these false positives revealed that

Fig. 3. Detection delay and false positives as a function of cache size for a fixed substring length of 250 bytes, distinct destination threshold of 10, sampling mask value of 0xf, and an offset limit of 100K. We observe that we have zero false positives for cache sizes less than 600 msec, while at the same time we have detection after about 10–20 attacks. We also observe that smaller cache sizes increase detection delay, and eventually prevent detection.

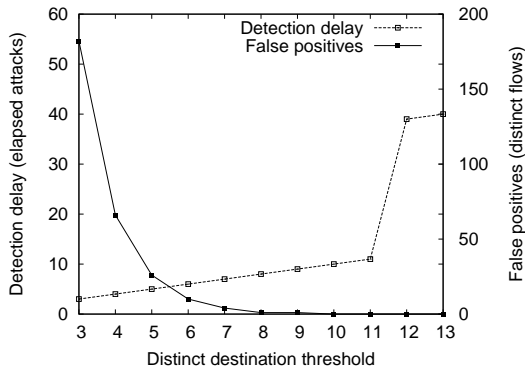
they are caused by peer-to-peer traffic and are encountered at random offsets in long-lived connections.

Summarizing, figures 2–5 suggest that our approach is able to identify the worms contained in the studied traces, without generating any false alarms.

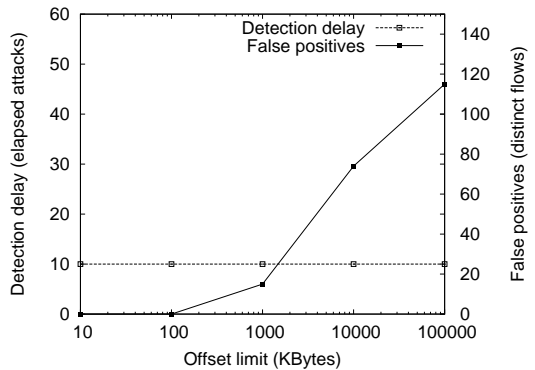
C. Performance

In our next set of experiments we evaluate the performance of our approach and measure the effects of fingerprint selection. In our experiments we have used deterministic sampling using the 0xf mask therefore the processed substrings have been reduced by a factor of 16. We carried out these experiments without an offset limit, to take into account the case where the smoother variation would be used as described in Section II-D.

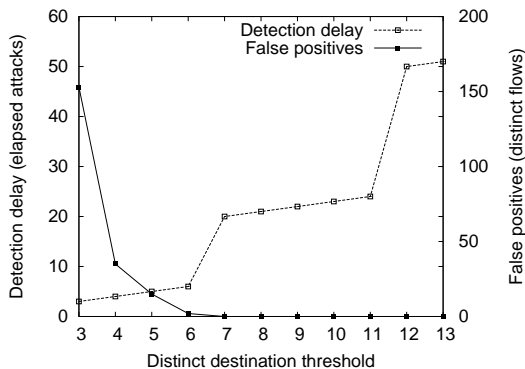
Table III shows the CPU time spent for the optimized and the unoptimized version of the worm detection system to process TRACE-I: a trace of network packets 5 Gbytes large. We see that the un-optimized version takes close to 13 minutes which corresponds to a processing rate of 57 Mbits/s,



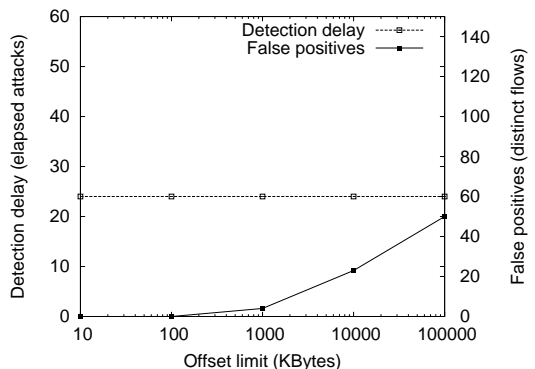
(a) TRACE-I



(a) TRACE-I



(b) TRACE-II



(b) TRACE-II

Fig. 4. Detection delay and false positives as a function of distinct destination threshold for a fixed substring length of 250 bytes, cache size of 500 msec, sampling mask value of 0xf, and offset limit of 100K. We observe that we have zero false positives for a threshold of 10 or greater, while at the same time we have detection after about 10–20 attacks.

Fig. 5. Detection delay and false positives as a function of offset limit for a fixed substring length of 250 bytes, cache size of 500 msec, distinct destination threshold value of 10, and sampling mask value of 0xf. We observe that for a limit of 100K or less, we have zero false positives, while at the same time we have detection after about 10–20 attacks.

TABLE III
CPU TIME CONSUMED BY EXPERIMENTS WITH TRACE-I AND
EXTRAPOLATED THROUGHPUT.

Sampling Mask	CPU Time	Throughput
0x0	13m	57 Mbits/s
0xf	145s	307 Mbits/s

while the optimized version takes close to 145 seconds, which corresponds to a processing rate of 307 Mbits/s.

Figure 6 demonstrates the savings in CPU-time that result from applying different sampling masks. We observe that performance increases exponentially with the number of bits in the sampling mask, as expected.

IV. RELATED WORK

Attempts to automatically detect unknown worms have been based on connection history patterns characteristic of worm spread, on popular packets or packet payloads, and on honeypots and correlation of data from distributed honeypots.

Moore et al. [9] focus on containment, as opposed to prevention and treatment, as an early response against worms,

and attempt to determine the minimum requirements posed on a system for it to successfully contain a worm. They argue that detecting and characterizing a worm is far easier than understanding the worm itself or the vulnerability being exploited and therefore containment can be applied very early on, during an epidemic. They lay out the design space for worm containment systems using the parameters of reaction time, containment strategy, and deployment strategy, and consider known and hypothesized worms using a combination of analytic modeling and simulation. They show that to prevent wide-spread infection in a 24 hour period, worms have to be detected within minutes of the start of an epidemic and nearly all of Internet paths, such as those covered by the 100 largest ASes, need to employ filtering.

Singh et al. [10] describe EarlyBird, a system for automatically detecting new worms based on highly repetitive packet content, an increased population of sources generating infections, and an increased number of destinations being targeted. They employ Rabin fingerprints [6] which allow incrementally computing the fingerprint of a sliding window of packet payload. They propose to first identify popular fixed-

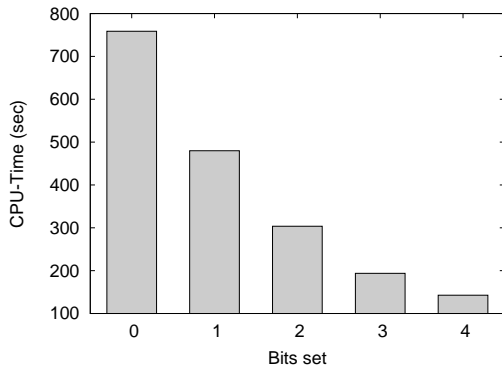


Fig. 6. CPU-time as a function of the number of bits set in the sampling mask.

length substrings using sample-and-hold [11] on the Rabin fingerprints and then use thresholds on distinct source and destination counts to ascertain whether the recurring content is a spreading worm. They further augment the incrementally computed Rabin fingerprints with the destination port number and argue that repetitive worm traffic always goes to the same ports while other repetitive content such as popular web pages or peer to peer traffic often goes to ports randomly chosen for each transfer.

There exist other network-level methods to detect worms that instead of packet contents rely on the effects of excessive scanning and probing, which is characteristic of worms encountered so far. For example, Bakos and Berk [12] propose an Internet-scale framework for worm detection that relies on ICMP destination unreachable (ICMP-T3) messages produced by failed connection attempts to identify worm activity and infected nodes. The system requires instrumented routers to forward such messages to a central collection point for analysis.

Weaver et al. [13] describe worm containment algorithms suitable for deployment in high-speed, low-cost network hardware based on detection of scanning worms. They divide the network into cells that communicate through worm containment devices and are quarantined in case of infection. The system relies on a low vulnerable host to probed host ratio to detect and contain a scanning worm at a faster rate than it can spread.

We should emphasize, however, that all methods based on scan detection provide good defense against scanning worms but are of limited use against hit-list worms or worms that discover targets without scanning.

Indra is a distributed security system that runs on top of a peer-to-peer network of Intrusion Detection Systems [14]. Indra correlates information gathered by individual IDSs in order to detect new types of attacks. Indra is very much work in progress, but based on the available information, we can say that it shares some goals with our approach.

Honeypots have also been proposed as a tool to detect and study attackers. Honeypots are ordinary computers that under normal circumstances provide no advertised services and have no ordinary users. Thus, honeypots should ordinarily have neither incoming nor outgoing traffic. If honeypots receive

or generate traffic, this traffic is immediately considered suspicious, and is probably the result of an interaction with an attacker. Honeycomb is a system that uses honeypots to detect intrusions and generate their signatures [15]. DOMINO [16] is another system that is based on correlation of data from geographically distributed honeypots and Intrusion Detection Systems.

The Autograph system [17] uses a first level screening test (scan detection in their evaluation), to identify suspicious traffic, and then counts substring popularity and number of sources to generate worm signatures. While superficially similar to Autograph, both EarlyBird and our system apply their heuristics directly on network traffic. The Autograph authors suggest that other mechanisms, such as EarlyBird, could be used as a first level screening test, enabling detection of hit-list worms.

From all described systems, our work is more closely related to the EarlyBird system [10]. Indeed, both approaches use frequently appearing fixed-length strings in network packet payloads as an indication for a new worm outbreak and they consider the number of distinct destinations where these strings are sent. However, there are also significant differences between our approach and the EarlyBird work:

- EarlyBird weights the same a string that appears in the first few kilobytes of a connection with a string that appears after hundreds of megabytes. On the other hand, we propose giving less weight to substrings appearing in connections that have already caused a great overhead to their initiator. This heuristic is especially effective against peer-to-peer control messages that appear like worm attacks, differing only in that they travel over already established, long-lived connections.
- EarlyBird augments the fingerprints with the destination port based on the observation that most worms target specific ports. Although this is a reasonable assumption, the Witty worm, with its random destination port, demonstrates an exception to this rule. On the other hand, we conservatively avoid relying on this criterion.
- By augmenting the fingerprints with the destination port, EarlyBird effectively ignores traffic sent from a server to a client (since the client's port is typically chosen at random), but it nevertheless has to compute and store fingerprints for such traffic. On the other hand, our system *explicitly* ignores traffic sent from a server to a client, using session tracking. Thus, apart from not having to augment fingerprints with the destination port, our system does not have to compute Rabin fingerprints at all for the bulk of the data transferred during a typical download operation (where the client sends a small request and receives an orders of magnitude larger response).
- As an additional mechanism to prevent false positives by contents of popular servers, EarlyBird also counts distinct sources. However, in some cases the number of sources in the start of an epidemic may appear significantly less than the number of targeted destinations. (With sequentially scanning worms, the first external source may probe and infect an entire network, before any other internal or external source appears.) We do not rely on the number

of sources in these experiments.

- EarlyBird uses a substring length of 40 bytes while we, on the other hand, justify a length of 150–250 bytes by showing that increasing the substring length has an adverse impact on false positives.
- The focus of the EarlyBird paper is on novel data-structures that facilitate efficient wire-speed worm detection. On the other hand, our detection heuristics allow us to skip much traffic and achieve similar performance results using far less optimized data-structures.

V. CONCLUSIONS AND FUTURE DIRECTIONS

We have demonstrated that signatures belonging to an Internet worm can be detected by finding strings with a high rate of transfer to different targets, and that false positives can be eliminated by considering strings of a reasonably large size appearing in connections generating reasonably small amounts of traffic.

We have shown that on the studied scale, of about 150 hosts, detection without false positives is possible with a detection delay that suggests a 7-14% infection. The detection was sensitive to worms generating collectively at least one attack about every 500 msec. These results are very encouraging. Presumably, higher aggregation would further reduce detection time, and also enable detection of less aggressive worms.

An important observation is that the thresholds required to trigger detection are relatively small, in our experiments eleven faked attacks would suffice for triggering a malicious false positive. We intend to solve this problem by using sampling of flows, which can be done efficiently at high speeds, to randomly choose sufficient amounts of traffic from a much larger pool of flows. This way, to pollute a certain fraction of the sampled flows, an attacker would have to generate much larger traffic, corresponding to that fraction over all flows in the entire pool.

Introducing further assumptions about worms can lead to even more detection heuristics that can be exploited, if applicable, to detect worms faster, with less false positives. For example, based on the observation that most worms spread using buffer overflow attacks, buffer overflow detection mechanisms can be employed to weed out false positives, or the system may process only substrings without the ASCII nul character, which must not be present in buffer overflows. We can further assume scanning, failed connections, etc. However, assumptions may also lead to false negatives. With this in mind, we intend to develop an opportunistic architecture that uses many focused detection heuristics to boost the sensitivity of the system, when possible, but at the same time is able to detect generic attacks, possibly with decreased sensitivity, allowing for graceful degradation of the system's effectiveness.

ACKNOWLEDGMENTS

This work was supported in part by the Greek General Secretariat for Research and Technology (GSRT) project EAR (USA-022) and ESTIA (04BEN8), a PAVET-NE project also funded by the GSRT. The work of K. Anagnostakis is also supported by OSD/ONR CIP/SW URI through ONR Grant N00014-04-1-0725. P. Akritidis and E. P. Markatos are also with the University of Crete. The work of K. Anagnostakis was done while at ICS-FORTH.

REFERENCES

- [1] C. Shannon and D. Moore, "The spread of the witty worm," 2004, <http://www.caida.org/analysis/security/witty/>.
- [2] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver, "The spread of the sapphire/slammer worm," CAIDA, ICSI, Silicon Defense, UC Berkeley EECS and UC San Diego CSE, Tech. Rep., 2003.
- [3] S. Staniford, V. Paxson, and N. Weaver, "How to Own the internet in your spare time," in *Proceedings of the 11th USENIX Security Symposium (Security '02)*, 2002.
- [4] S. Staniford, D. Moore, V. Paxson, and N. Weaver, "The top speed of flash worms," in *WORM '04: Proceedings of the 2004 ACM workshop on Rapid malware*. ACM Press, 2004, pp. 33–42.
- [5] N. T. Spring and D. Wetherall, "A protocol-independent technique for eliminating redundant network traffic," in *Proceedings of the conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*. ACM Press, 2000, pp. 87–95.
- [6] M. Rabin, "Fingerprinting by random polynomials," Center for Research in Computing Technology - Harvard University, Tech. Rep. 15-81, 1981.
- [7] M. Roesch, "Snort: Lightweight intrusion detection for networks," in *Proceedings of USENIX LISA '99*, November 1999, (software available from <http://www.snort.org/>).
- [8] U. Manber, "Finding similar files in a large file system," in *Proceedings of the USENIX Winter 1994 Technical Conference*, San Francisco, CA, USA, Jan. 1994, pp. 1–10.
- [9] D. Moore, C. Shannon, G. M. Voelker, and S. Savage, "Internet quarantine: Requirements for containing self-propagating code," in *INFOCOM*, 2003.
- [10] S. Singh, C. Estan, G. Varghese, and S. Savage, "Automated worm fingerprinting," in *Proceedings of the ACM/USENIX Symposium on Operating System Design and Implementation*, Dec. 2004.
- [11] C. Estan and G. Varghese, "New directions in traffic measurement and accounting," in *Proceedings of the 2001 ACM SIGCOMM Internet Measurement Workshop*, 2001, pp. 75–80.
- [12] G. Bakos and V. Berk, "Early detection of internet worm activity by metering icmp destination unreachable messages," in *Proceedings of the SPIE Aerosense 2002*, 2002.
- [13] N. Weaver, S. Staniford, and V. Paxson, "Very fast containment of scanning worms," in *USENIX Security Symposium*, 2004, pp. 29–44.
- [14] R. Janakiraman, M. Waldvogel, and Q. Zhang, "Indra: A peer-to-peer approach to network intrusion detection and prevention," in *Proceedings of IEEE WETICE 2003*, June 2003.
- [15] C. Kreibich and J. Crowcroft, "Honeycomb – creating intrusion detection signatures using honeypots," in *Proceedings of the Second Workshop on Hot Topics in Networks (HotNets-II)*. Cambridge, Massachusetts: ACM SIGCOMM, Nov. 2003.
- [16] V. Yegneswaran, P. Barford, and S. Jha, "Global intrusion detection in the DOMINO overlay system," in *Proceedings of the 11th Annual Network and Distributed System Security Symposium*, San Diego, CA, Feb. 2004.
- [17] H.-A. Kim and B. Karp, "Autograph: Toward automated, distributed worm signature detection," in *Proceedings of the 13th Usenix Security Symposium*, 2004.